Korešpondenčný seminár z programovania
XXXIV. ročník, 2016/17
Katedra základov a vyučovania informatiky FMFI UK,
Mlynská Dolina, 842 48 Bratislava

Návody k úlohám 1. kola zimnej časti kategórie T

V kategórii T neuvádzame vzorové riešenia ale skôr návody na riešenie úloh. Ich cieľom je prezradiť vám hlavnú myšlienku riešenia, aby ste podľa návodu mohli riešenie domyslieť sami. Občas teda vynecháme niektoré drobné detaily, neuvádzame implementácie dátových štruktúr a nerozpisujeme niektoré kroky.

Neuvádzame ani vzorové programy, pretože chceme, aby ste po prečítaní návodu naprogramovali tie úlohy, ktoré ste nespravili počas trvania série. Keď si tieto riešenia sami naprogramujete, naučíte sa tým oveľa viac ako pozeraním sa na zdrojové kódy.

Implementácie všeobecne známych algoritmov a dátových štruktúr môžete nájsť vo vzorových riešeniach kategórií Z a O starších ročníkov KSP alebo aj na internete.

1. Tabuľa a fixka

návod písal mišof
(max. 0 b za popis, 20 b za program)

Začneme zopár zjavnými pozorovaniami. V prvom rade, ak má byť červený nejaký pixel, cez ktorý fixkou nikdy nejdeme, riešenie zjavne neexistuje. A ako to bude s ostatnými pixelmi? Pre každý z nich je dôležitý len jeden okamih: ten, kedy cez neho fixka prechádza poslednýkrát. Bez ohľadu na to, akej farby bol dovtedy, odteraz až do konca bude červený, resp. biely podľa toho, či fixka ešte píše alebo už nie.

Na základe týchto pozorovaní vieme spraviť pomalé riešenie. To bude vyzerať nasledovne:

- Vyrobíme si v pamäti bitmapu zodpovedajúcu tabuli.
- Postupne krok po kroku simulujeme pohyb fixky a pre každé políčko si zistíme posledný timestamp, kedy sme tam boli.
- Ak existuje políčko, ktoré má byť červené, ale sme tam nikdy neboli, riešenie neexistuje.
- Pre každé políčko, ktoré má skončiť červené, sa pozrieme na posledný timestamp, kedy sme tam boli. V čase t_1 rovnom maximu týchto timestampov musela fixka ešte stále písať.
- Pre každé políčko. ktoré má skončiť biele a cez ktoré sme aspoň raz išli fixkou, sa tiež pozrieme na posledný timestamp, kedy sme tam boli. V čase t_2 rovnom minimu týchto timestampov musí byť už fixka vyschnutá.
- Riešenie existuje práve vtedy, keď $t_1 < t_2$.

Rýchlejšie riešenie

Prečo je vyššie uvedené riešenie pomalé? Preto, že obmedzenia v zadaní boli trochu zákerné. Najhoršie vstupy vyzerajú tak, že spravíme 10^6 ťahov fixkou a každý z nich prejde až rádovo 10^6 políčok. Dokopy by sme teda v predchádzajúcom riešení potrebovali simulovať až 10^{12} krokov, a to už je priveľa.

Ako toto riešenie zrýchliť? Každý pohyb fixkou je pohybom buď v niektorom riadku alebo v niektorom stĺpci. Roztriedime si teda pohyby fixkou na kôpky podľa toho, v ktorom riadku/stĺpci vedú. Následne spracujeme každý riadok a stĺpec zvlášť. Zakaždým budeme robiť to isté: spočítame, aké číslo by bolo na ktorom jeho políčku, keby existovali len tie "jeho" ťahy fixkou.

Spracovanie jedného riadku/stĺpca sa dobre implementuje pomocou zametania. Pre jednoduchosť predpokladajme, že spracúvame riadok. To, čo máme na vstupe, je množina intervalov. Každý interval vieme popísať štyrmi číslami: v ktorom stĺpci začína, v ktorom stĺpci končí, akým číslom (timestampom) začína a či čísla zľava doprava rastú alebo klesajú.

Usporiadame si začiatky aj konce všetkých intervalov. Potom prechádzame riadkom zľava doprava, pričom si aktuálne intervaly pamätáme napr. v sete (usporiadanej množine) podľa čísla, ktorým začínajú. (Tu sa oplatí rozmyslieť si, že intervaly sú disjunktné, takže ak má v nejakom bode jeden interval väčšiu hodnotu ako iný, má ju väčšiu aj všade inde.) Na každé políčko riadku zapíšeme najväčšiu z hodnôt aktuálnych intervalov.

Ak riadok obsahuje k intervalov, spracujeme ho takto v čase $O(k \log k + c)$, kde c je počet stĺpcov. Dokopy teda celé riešenie zbehne v čase $O(rc + n \log n)$.

2. Trojuholníková nerovnosť

návod písal Buj
(max. 0 b za popis, 20 b za program)

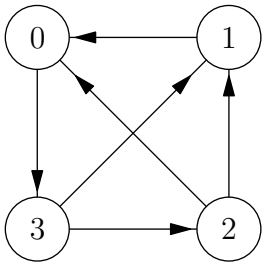
Pri pohľade na túto úlohu každému padne do oka jej netradičné bodovanie. Aj keď ste nenašli najlepšiu

stratégiu pre Miša, mohli ste získať za úlohu body. Neznamená to ale, že je ťažké nájsť najlepšiu stratégiu – ide ju spočítať *exaktne*, a netreba sa uchýliť k heuristickým metódam, magickým konštantám, a podobným.

Ako hrá Žaba?

Máme maticu A rozmerov $n \times n$, v ktorej sú hodnoty 0, $\frac{1}{2}$ alebo 1 podľa zadaného grafu, pričom hodnotu $\frac{1}{2}$ máme iba na diagonále. V každom kole si Mišo tajne vyberie riadok r a Žaba si tajne vyberie s. Následne sa ich voľby odkryjú, a podľa hodnoty v riadku r a stĺpci s sa pridelia body. Mišo získa toľko bodov, koľko je tá hodnota. Žaba vždy získa taký počet bodov, aby celkový počet udelených bodov bol 1.

	0	1	2	3
0	0.5	1	1	0
1	0	0.5	1	1
2	0	0	0.5	1
3	1	0	0	0.5



Mišo si zvolil pravdepodobnosti $p_1, \ldots, p_n$ – p_i reprezentuje pravdepodobnosť, s akou si zvolí i-ty riadok. Súčet pravdepodobností je 1. Predstavme si, že si Žaba zvolil stĺpec s. Potom s pravdepodobnosťou p_1 zvolil Mišo riadok 1, a v tom prípade by získal $A_{1,s}$ bodov. S pravdepodobnosťou p_2 zvolil riadok 2, a získal by vtedy $A_{2,s}$ bodov. Podobne pre ostatné prípady. V priemere by tak získal toľkoto bodov:

$$p_1 \cdot A_{1,s} + p_2 \cdot A_{2,s} + \ldots + p_n \cdot A_{n,s} = (p_1, p_2, \ldots, p_n) \cdot (A_{1,s}, A_{2,s}, \ldots, A_{n,s})$$

Samozrejme, Žaba zvolí s tak, aby v priemere získal Mišo čo najmenej bodov. Môžeme sa na to pozerať tak, že Mišo priradil riadkom pravdepodobnosti, a následne každý riadok matice prenásobil pravdepodobnosťou, ktorú riadku priradil. Následne príde Žaba, a vyberie si stĺpec s najmenším súčtom. Tento súčet reprezentuje očakávaný počet bodov, ktoré získa Mišo.

0.3		0.5	1	1	0		0.15	0.3	0.3	0
0.2	×	0	0.5	1	1	=	0	0.1	0.2	0.2
0.4		0	0	0.5	1		0	0	0.2	0.4
0.1		1	0	0	0.5		0.1	0	0	0.05
							0.25	0.4	0.7	0.65

(Napríklad Mišo si zvolil pravdepodobnosti 0.3, 0.2, 0.4, 0.1. Potom vie Žaba hrať tak, že Mišo získa v priemere 0.25 bodov – vždy zvolí sivý stĺpec.)

Nash equilibrium

Pod *stratégiou* hrania tejto hry budeme rozumieť pravdepodobnosti, s akými si zvolíme vrcholy grafu. Zoberme si dvoch hráčov, prvý nech má stratégiu $P = (p_1, p_2, \ldots, p_n)$. Druhý nech má stratégiu $Q = (q_1, q_2, \ldots, q_n)$.

Hovoríme, že (P, Q) sú v Nashovom equilibriu, ak žiaden z hráčov nevie zlepšiť svoju stratégiu proti tomu druhému za predpokladu, že ten druhý svoju stratégiu nezmení. Inak povedané, žiadnemu z hráčov sa neoplatí meniť svoju stratégiu – nezlepší si tým priemerný počet získaných bodov.

Známa veta pána Nasha hovorí, že naša hra má Nashovo equilibrium – existujú teda dve stratégie P, Q také, že sa žiadnemu hráčovi neoplatí meniť stratégiu. Potom musia mať obaja hráči ale očakávaný počet získaných bodov 0.5 – každý z hráčov sa totižto môže rozhodnúť skopírovať súperovu stratégiu, a tým by dosiahol zisk presne 0.5. My ale vieme, že si týmto nemohol polepšiť (lebo to je Nashovo equilibrium) – takže už predtým musel mať zisk aspoň 0.5. Celkový zisk je ale 1 – musí to byť preto presne 0.5.

To ale znamená, že Mišo vie priradiť riadkom také pravdepodobnosti, aby mal zisk v najhoršom prípade 0.5.

Ako nájsť equilibrium?

Niektorí ste sa možno dopátrali ku komplikovaným algoritmom ako Simplex Algorithm alebo Lemke-Howson Algorithm. S našimi obmedzeniami ($n \leq 9$) vieme problém vyriešiť aj jednoduchšie.

Predpokladajme, že Mišova stratégia P je najlepšia – teda má v najhoršom prípade zisk 0.5. Potom musí tvoriť dvojica (P, P) Nash equilibrium. Zafixujme stratégiu prvého hráča, a pozrime sa na to, ako by mohol druhý hráč zlepšiť svoj zisk odklonom od stratégie P. Povedzme, že by si v kole vybral stĺpec s. Potom očakávaný zisk je

$$(p_1, p_2, \ldots, p_n) \cdot (A_{1,s}, A_{2,s}, \ldots, A_{n,s})$$

a je nanajvýš 0.5, lebo P je najlepšia stratégia. Ak to nie je 0.5, musí to byť menej ako 0.5. Potom ale P nikdy nemôže hrať tento vrchol s – ak by ho hral, tak by mal celkový zisk menší ako 0.5. A to nemôže, lebo keď máme proti sebe dve identické stratégie, tak zisk oboch hráčov musí byť 0.5.

Dostávame tak kľúčové pozorovanie – pre každé $s \in \{1, 2, \ldots, n\}$ platí, že buď $p_s = 0$, alebo

$$(p_1, p_2, \ldots, p_n) \cdot (A_{1,s}, A_{2,s}, \ldots, A_{n,s}) = 0.5$$

Môžeme preto postupovať takto: pre každý vrchol si tipneme, ktorá z týchto možností nastala. Dostaneme tak dve množiny vrcholov – prvá množina obsahuje vrcholy, ktorým priradíme pravdepodobnosť výberu 0. Druhá množina obsahuje tie vrcholy s, ktoré spĺňajú

$$(p_1, p_2, \ldots, p_n) \cdot (A_{1,s}, A_{2,s}, \ldots, A_{n,s}) = 0.5$$

Týchto rovníc máme k, kde k je počet vrcholov v druhej množine. Máme teda k rovníc o k neznámych (lebo pre $n - k$ vrcholov z prvej množiny platí $p_i = 0$) – na vyriešenie použijeme Gaussovu elimináciu. Takto ale iba získame kandidáta, nemusí to byť ešte hľadaná najlepšia stratégia (vieme ale, že niektorá z nich bude najlepšia). Preto na záver musíme ešte overiť

1. Či sú vypočítané hodnoty naozaj pravdepodobnosti, teda či sú aspoň 0 a najviac 1. (Môže sa nám totiž stať, že nám vyjdu hodnoty niektorých premenných záporné alebo väčšie ako 1.)
2. Či to je naozaj najlepšia stratégia. To znamená, že nech si protivník zvolí ľubovoľný z vrcholov $1, 2, \ldots, n$, v každom prípade máme zisk zisk aspoň 0.5.

Časová zložitosť je $O(2^n \cdot n^3)$ – máme 2^n tipov, a v každom robíme Gaussovu elimináciu na matici s $k \leq n$ rovnicami a k neznámymi.

návod písal Buj

3. Tristo

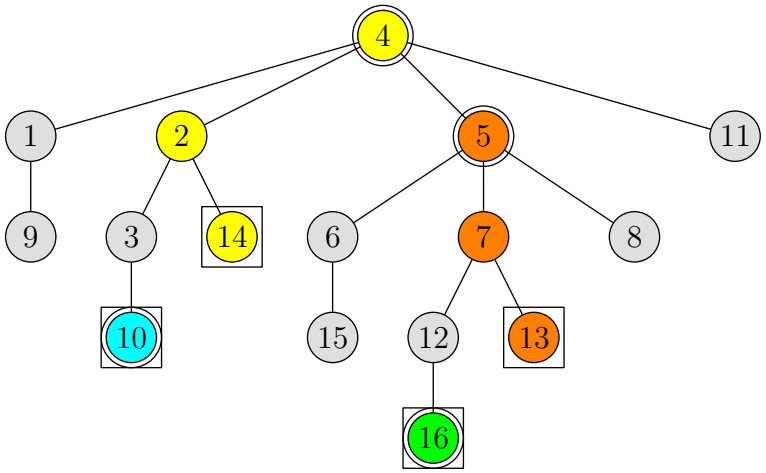
(max. 0 b za popis, 20 b za program)

Najprv sa zamyslime nad tým, čím je určený stav hry. Napríklad si môžeme pamätať popis stromu, a pre každú dážďovku všetky vrcholy, v ktorých je niektorý jej článok. Potrebujeme to ale? V skutočnosti nám stačí si pamätať iba vrchol, v ktorom je hlava dážďovky, a pozíciu chvostu. Tým je zvyšok tela jednoznačne určený.

Keď hráč klikne na dážďovku, chceme vedieť efektívne zistiť, kde sa po kliknutí bude nachádzať jej hlava a kde jej chvost. Z pohľadu kliknutej dážďovky nastane jedna z dvoch možností:

1. Je nad nami nejaká dážďovka, na ktorú cestou hore narazíme. Konkrétne narazíme na takú dážďovku, ktorej hlava vrchol je (nie nutne priamym) predkom nášej hlavy, a spomedzi všetkých takých dážďoviek je najbližšie.
2. Nie je nad nami žiadna iná dážďovka, a komplex opustíme. To nastane vtedy, keď žiaden predok našej hlavy neobsahuje hlavu inej dážďovky. Tento prípad je jednoduchý, a ďalej sa ním zaoberať nebudeme.

V prvom prípade nás zaujíma, o koľko hore sa posunie naša hlava. O rovnako veľa sa posunie aj náš chvost. Keď už ale poznáme dážďovku, do ktorej narazíme, vieme ľahko zistiť, kde na ňu narazíme – v najnižšom spoločnom predkovi našej hlavy a jej chvostu. Z toho, aká je vzdialenosť našej hlavy od koreňa (*hĺbka*), a aká je hĺbka "miesta narazenia" vieme ľahko vypočítať, o koľko sa posunieme hore. Nakoniec musíme našu hlavu aj chvost o toľko posunúť hore.



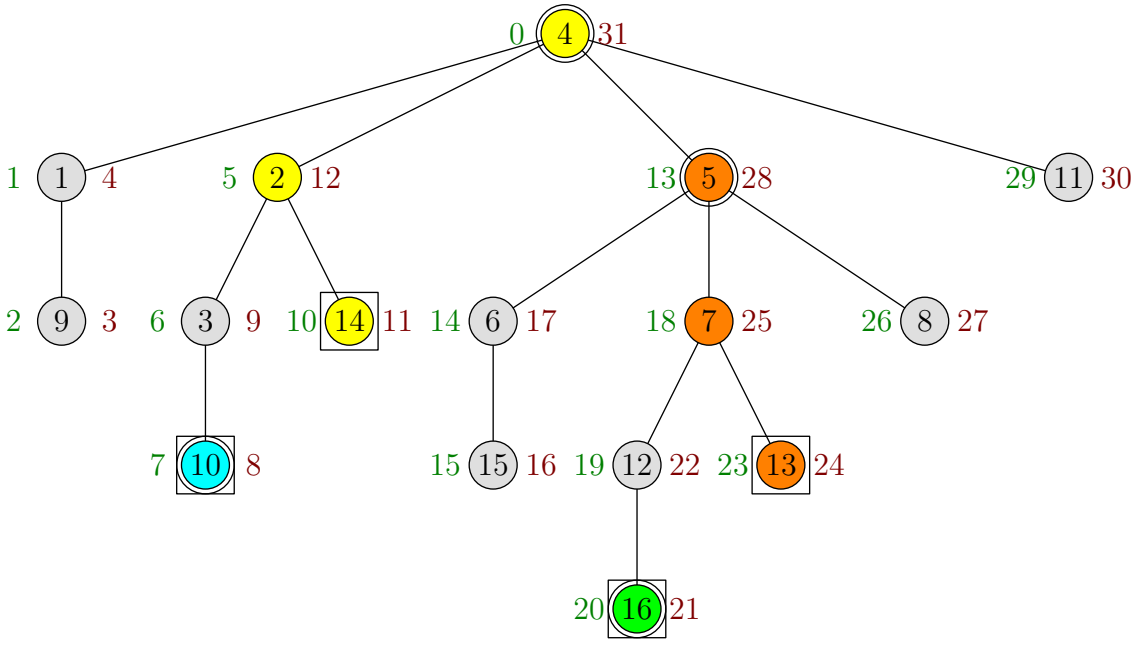
(Povedzme, že sme klikli na zelenú dážďovku. Jej hlava je na 16, najbližší predok s hlavou je 5 – takže narazime do červenej dážďovky. Jej chvost je na 13, a najnižší spoločný predok 13 a 16 je 7 – tu narazíme. Takže sa posunieme hore o $hlbka_{16} - hlbka_7 - 1 = 4 - 2 - 1 = 1$.)

Chceme vedieť robiť efektívne tieto tri operácie:

1. Dostaneme vrchol u = hlava kliknutej dážďovky. Chceme nájsť jeho najbližšieho predka s hlavou.
2. Zistíme jeho majiteľa, a nemu prislúchajúci chvostový vrchol v. Hľadáme najbližšieho spoločného predka u a v, označme ho x.
3. Dostaneme $d = hlbka_u - hlbka_x - 1$, a vrchol y (naša hlava alebo chvost). Hľadáme d-teho predka y.

Ako spraviť tieto operácie rýchlo

Prvú operáciu vieme vykonať nasledovne: každému vrcholu stromu priradíme *čas objavenia* (kedy sme ho v prehľadávaní do hĺbky začali spracovávať) a *čas opustenia* (kedy sme s jeho spracovávaním skončili). Takto každé z čísel $1, 2, \ldots, 2n$ prislúcha jednému vrcholu – buď ako čas objavenia, alebo ako čas opustenia.

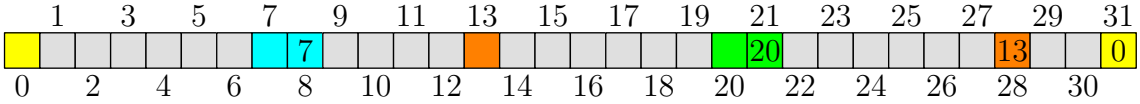


(Zelené číslo uvedené naľavo od vrcholu je jeho čas objavenia, červené číslo uvedené napravo je čas opustenia.)

Keď chceme nájsť najbližšieho hlavového predka vrcholu v, tak v podstate spomedzi všetkých neskôr opustených hlavových vrcholov hľadáme najskorší taký, že sme ho objavili skôr ako v.

V poli dĺžky $2n$ si na pozícii i budeme pamätať, či existuje hlavový vrchol s koncovým časom i. Ak áno, tak na tej pozícii bude čas objavenia tohto vrchola. Potom dotaz "zisti najbližšieho hlavového predka vrcholu v" vieme zodpovedať takto: pozrieme sa na všetky pozície v poli väčšie ako čas opustenia v. Nájdeme najmenšiu

pozíciu, na ktorej je hodnota menšia ako čas objavenia v. Táto pozícia zodpovedá času opustenia najbližšieho hlavového predka v.



Prechod celým poľom by mal časovú zložitosť až $O(n)$, dá sa to ale implementovať efektívne pomocou intervalového stromu v čase $O(\log n)$. Samozrejme treba túto dátovú štruktúru udržiavať – vždy, keď sa nejaká hlava presunie, ju vhodne upravíme.

Druhá operácia je štandardný *lowest common ancestor*.

Tretia operácia: rozdelíme vrcholy stromu do vrstiev podľa ich hĺbky. Na každej vrstve vrcholy usporiadame podľa ich začiatočných časov. Keď hľadáme d-teho predka vrcholu y, pozrieme sa na vrstvu $hlbka_y - d$. Vďaka začiatočným časom vieme binárne vyhľadať predka y – je to posledný vrchol na tej vrstve, ktorý sme začali spracovávať pred y. (Táto operácia je popísaná aj v nedávnom vzorovom riešení 7. úlohy 1. kola 34. ročníka KSP-O.)

Keď takto implementujeme všetky tri operácie, dostaneme riešenie, ktoré vie odsimulovať jedno kliknutie v čase $O(\log n)$.

návod písal Buj

4. Taký neporiadok!

(max. 0 b za popis, 20 b za program)

Na začiatok sa zbavme nezaujímavého prípadu: ak neexistuje žiaden kôš, tak Kubo vo svojej misii neuspeje. Vypíšeme preto `-1`. Ďalej predpokladáme existenciu koša.

Kubove upratovanie vyzerá nasledovne – najprv sa presunie k nejakému odpadku, a zdvihne ho. Potom sa presunie ku košu, a odpadok hodí do koša. Toto sa opakuje, kým existujú odpadky.

Zoberme si ľubovoľný odpadok. Zaujímavé sú najlbližší kôš naľavo (L), a najbližší kôš napravo (R). Zrejme nemá zmysel hádzať ten odpad do niektorého iného koša – cestou k nemu by sme ho vedeli zahodiť do L alebo R.

Dostávame jednoduché riešenie v časovej zložitosti $O(n! \cdot 2^n)$, kde n je počet odpadkov (nie košov) – vyskúšame všetky možné poradia, v ktorých mohol Kubo zahodiť odpadky, a pre každý odpadok máme dve možnosti, kam ho mohol zahodiť. Toto riešenie bolo hodnotené 6 bodmi.

Typy odpadkov

Pre jednoduchosť zatiaľ predpokladajme, že Kubo začína na pozícii, kde sa nachádza kôš. Potom pre každý odpadok vieme sledovať Kuba, ako ho zahadzuje. To pozostáva z dvoch častí – cesta od koša k odpadku, a cesta od odpadku k (nie nutne tomu istému) košu. Pre každý odpadok máme preto štyri možnosti, ako môže vyzerať jeho cesta do koša.

1. Kubo začne v koši, ktorý je najlbližšie naľavo, a odpadok hodí do toho istého koša. Teda Kubo začne aj skončí v koši L.
2. Kubo začne aj skončí v najbližšom koši napravo od odpadku (R).
3. Kubo začne v L a skončí v R.
4. Kubo začne v R a skončí v L.

Podľa týchto štyroch prípadov budeme rozlišovať odpadky typu 1, 2, 3 a 4. Všimnime si, že keď Kubo navštívi akýkoľvek kôš, tak môže rovno poupratovať všetky odpadky typu 1 a 2, ktoré sa k tomu košu vzťahujú.

Načo 3 a 4?

Povedzme, že máme odpadok, ktorý je od ľavého koša vzdialený l a od pravého koša r. Ak je typu 1, tak poupratovať ho nás bude stáť $2l$ času, pre typ 2 zas $2r$ času. V prípade 3 a 4 nás bude stáť $l + r$ času. Ľahko vidíme, že ak by sme sa na to pozerali iba takto, tak prípady 3 a 4 sa vôbec neoplatia – vždy je jeden z prípadov 1 alebo 2 aspoň rovnako časovo dobrý, ak nie lepší.

Prípady 3 alebo 4 majú ale jednu výhodu – umožňujú nám presúvať sa medzi košmi. Bez nich by sme sa museli presúvať tak, že by sme pri presune nezahodili žiaden odpadok, čo by bola strata času.

Oplatí sa prechádzať tam a späť veľakrát?

Každé dve susedné koše nám určujú *úsek odpadkov*. Zrejme ak je najľavejší úsek prázdny, môžeme ho ignorovať. Podobne pre najpravejší úsek. Budeme ďalej predpokladať, že posledné úseky v oboch smeroch obsahujú aspoň jeden odpadok.

Zamyslime sa nad tým, či sa nám vôbec niekedy oplatí prejsť úsekom viac ako dvakrát. Napríklad prvýkrát zľava doprava, potom sprava doľava a nakoniec zľava doprava. To znamená, že Kubove upratovanie vyzerá nasledovne.

1. Kubo upratuje a časom sa dostane ku košu L.
2. Kubo sa presunie z L do R. Pritom možno zahodí odpadok typu 3.
3. Kubo upratuje a časom sa dostane späť ku košu R.
4. Kubo sa presunie z R do L. Pritom možno zahodí odpadok typu 4.
5. Kubo upratuje a časom sa dostane späť ku košu L.
6. Kubo sa presunie z L do R. Pritom možno zahodí odpadok typu 3.
7. Kubo upratuje ďalej.

Namiesto takéhoto postupu mohol zvoliť ale tento, efektívnejší: spraví 1 a 5, potom spraví 2, a nakoniec spraví 3 a 7.

Pritom sme ale vynechali dve odpadky, ktoré Kubo zahodil v 4 a 6. Každý z nich ale vieme zahodiť do toho koša, ku ktorému je bližšie:

Ak je odpadok bližšie k L ($l \leq r$), tak Kubo ho zodvihne a zahodí v prvej časti upratovania (**pred presunom** doprava). Toto nás stojí čas $2l$, čo je aspoň tak dobré, ako v pôvodnom upratovacom pláne – tam nás to stálo $l + r$.

Ak je odpadok bližšie k R, tak ho Kubo zodvihne a zahodí v druhej časti upratovania (**po presune** doprava).

Vidíme teda, že nemá zmysel prechádzať viac ako dvakrát. Takže každý úsek prejdeme buď 2-krát, 1-krát, alebo 0-krát (začneme aj skončíme na jednom konci, bez toho, aby sme niekedy prešli na druhú stranu).

Kubove možnosti

Z horeuvedeného vyplýva, že Kubovo upratovanie musí vyzerať naslovne.

1. Začne pri koši, potom jedným smerom prejde všetky úseky v tom smere.
2. V poslednom z tých úsekov sa otočí – nemusí ho teda prejsť celý. (Predchádzajúce úseky musel prejsť celé, inak by nemohol pokračovať v smere. Za posledným úsekom ale už nič nie je, preto ho nemusí prejsť celý.)
3. Prejde všetky úseky v druhom smere.
4. Posledný úsek nemusí prejsť celý, z rovnakých dôvodov, aké sú uvedené v druhom kroku.

Máme teda štyri triedy úsekov:

1. Najľavejší úsek (ak začiatočný kôš nie je úplne vľavo).
2. Iné úseky naľavo od začiatočného koša.
3. Najpravejší úsek (ak začiatočný kôš nie je úplne vpravo).
4. Iné úseky napravo od začiatočného koša.

Kubovo upratovanie každej triede priradí typ podľa toho, akým spôsobom tento úsek prejde. Typy sú nasledovné:

1. Úsek bol prejdený práve raz (niektorým smerom, nezáleží na tom, ktorým). Pre takýto úsek vieme ľahko zistiť najlepší spôsob, akým pozbierať všetky odpadky v ňom – všetky odpadky okrem jedného budú typu 1 alebo 2, a posledný odpadok bude použitý na presun. Bude teda typu 3 alebo 4, a je zvolený tak, aby trvalo upratanie úseku čo najkratšie.

2. Úsek bol prejdený dvakrát. Najlepšie možné upratanie vieme zistiť podobne, ako v predchádzajúcom prípade, až na to, že až dve odpadky použijeme na presun.

3. Úsek nebol prejdený ani raz, a navštívili sme preto iba jeho ľavý koniec. Všetky odpadky v úseku musia byť typu 1, a vieme preto zistiť celkový čas potrebný na upratovanie.

4. Úsek nebol prejdený ani raz, a navštívili sme iba jeho pravý koniec. Všetky odpadky v úseku musia byť typu 2, a ľahko spočítame celkový čas potrebný na upratovanie.

Možností, ako môže Kubovo upratovanie vyzerať, je 8 – určujeme iba smer, ktorým sa na začiatku vydá, typ najľavejšieho úseku a typ najpravejšieho úseku. Pre každú možnosť vieme zistiť trvanie upratovania v čase $O(n)$.

Čo keď Kubo nezačína pri koši?

Začiatok upratovania je nasledovný – Kubo sa presunie k odpadku, ktorý je v rovnakom úseku ako on, potom sa presunie k niektorému koncu úseku, a odpadok hodí do koša. Tým zahodil jeden odpadok a presunul sa k niektorému košu – od tohto momentu by sme vedeli použiť predchádzajúci postup.

Vyskúšame dve možnosti podľa toho, do ktorého z krajných košov odpadok zahodí. Zahodenie jedného odpadku v začiatočnom úseku ovplyvní iba cenu (čas potrebný na upratanie) tohto úseku. Vyskúšame všetky možné voľby prvého zahodeného odpadku – v každej voľbe vieme rýchlo v čase $O(1)$ vypočítať vplyv na čas potrebný na upratanie úseku. Spomedzi všetkých možných volieb vyberieme tú cenovo najlacnejšiu.

Tento (všeobecný) prípad časovú zložitosť riešenia neovplyvní – tá ostane $O(n)$.

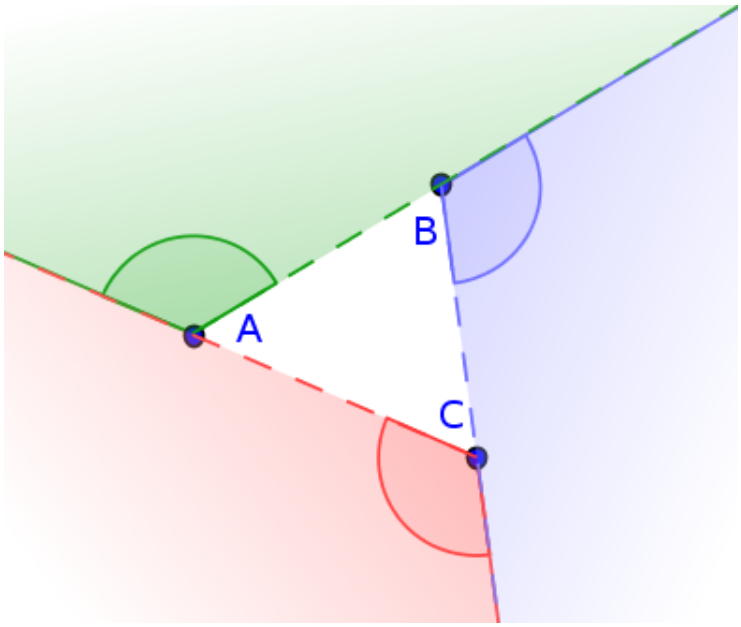
návod písal Baklažán

5. Trojuholníky a body

(max. 0 b za popis, 20 b za program)

Naše riešenie založíme na princípe inklúzie a exklúzie (alebo ak chcete, zapojenia a vypojenia). Namiesto toho, aby sme rátali body v zadanom trojuholníku, zrátame vždy body ležiace mimo trojuholníku a ich počet odrátame od celkového počtu bodov n.

Časť roviny ležiacu mimo trojuholníka ABC vieme rozdeliť na tri uhly nasledovným spôsobom:



Ak si navyše povieme, že každý uhol obsahuje body ležiace na jeho „ľavom'‘ramene, ale neobsahuje body ležiace na jeho „pravom" ramene, ani bod ležiaci v jeho vrchole [1], bude mať naša trojica uhlov jednu dobrú vlastnosť: Každý bod roviny ležiaci mimo trohuholníka ABC leží v práve jednom z našich troch uhlov.

To znamená, že ak chceme spočítať body v našom trojuholníku, stačí nám spočítať počty bodov ležiacich v našich troch uhloch a tieto tri čísla odrátať od čísla n. Body ležiace v uhle budeme počítať nasledovne:

- Každý bod P, ktorý sme dostali na vstupe, spojíme myslenými polpriamkami s každým iným bodom na vstupe. Tieto polpriamky následne utriedime podľa uhla. Toto nám stačí urobiť pre každý bod raz na začiatku. Celkovo nám to teda zaberie čas $O(n^2 \log n)$.
- Vždy, keď nás bude zaujímať počet bodov v nejakom uhle s vrcholom v P, binárne vyhľadáme, kde by v našom utriedenom zozname polpriamok boli ramená tohto uhla. Počet polpriamok ležiacich medzi ramenami je počet bodov ležiacich v uhle. Počet bodov ležiacich v uhle teda vieme zistiť v čase $O(\log n)$.

Celé riešenie teda vyzerá nasledovne:

1. Po načítaní vstupu si pre každý bod predrátame zoznam polpriamok vychádzajúcich z daného bodu, utriedený podľa uhla.

[1] takéto chápanie uhlov je analogické s používaním polootvorených intervalov

2. Následne spracúvame jednotlivé otázky, nasledovným spôsobom:
    1. Časť roviny ležiacu mimo zadaného trojuholníka si rozdelíme na tri uhly.
    2. Pre každý uhol vypočítame počet bodov v ňom (to nám bude trvať čas $O(\log n)$.
    3. Tieto tri čísla odrátame od celkového počtu bodov n a výsledok vypíšeme.

Časová zložitosť nášho algoritmu je teda $O((n^2 + q) \log n)$. Pamäťová zložitosť je $O(n^2)$.

Implementačné zákernosti

Pri implementácii si bolo treba dať dobrý pozor na nasledujúce detaily:

- Správnu implementáciu triedenia a binárneho vyhľadávania podľa uhla (keďže na zoznam polpriamok sa treba pri vyhľadávaní pozerať ako na cyklický zoznam).
- Presnosť výpočtov. Ideálne bolo robiť všetko v celočíselnej aritmetike s využitím vektorových súčinov. V niektorých vstupoch boli trojice bodov, ktoré ležali takmer na priamke. Implementácia spoliehajúca sa na goniometrické funkcie a aritmetiku reálnych čísel s nimi mohla mať problém.